

DirectX10: porting, performance and “gotchas”

Guennadi Riguer
AMD

**TAKE
CONTROL**
www.gdconf.com

MOSCONE
CENTER



DirectX 10 and Me...

- ⊕ The best DirectX ever – I love it!
- ⊕ Powerful feature set
 - ⊕ With power comes responsibility
 - ⊕ More ways to do something wrong and kill performance
- ⊕ New or changed behaviors – many porting “gotchas”



Deprecated Features

- ⌘ Alpha test
- ⌘ Triangle fans
- ⌘ Point sprites
- ⌘ Wrap texture modes
- ⌘ TnL clip planes



Fullscreen Initialization

- ⊕ Special flag in swap chain description to allow mode switch

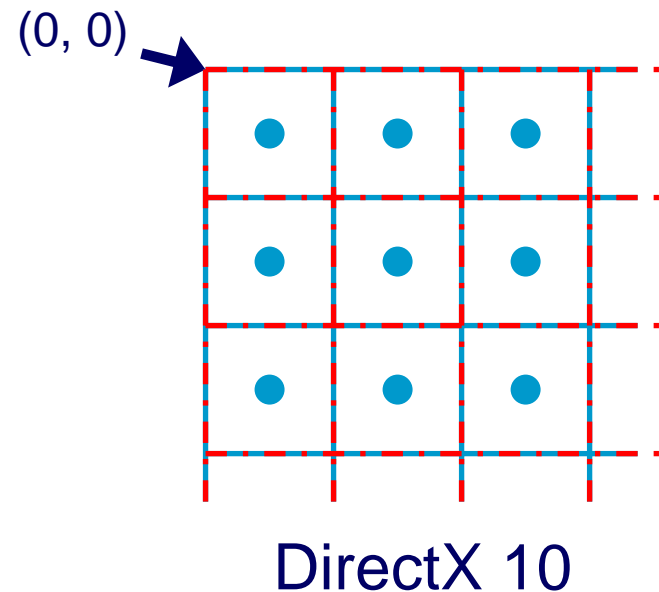
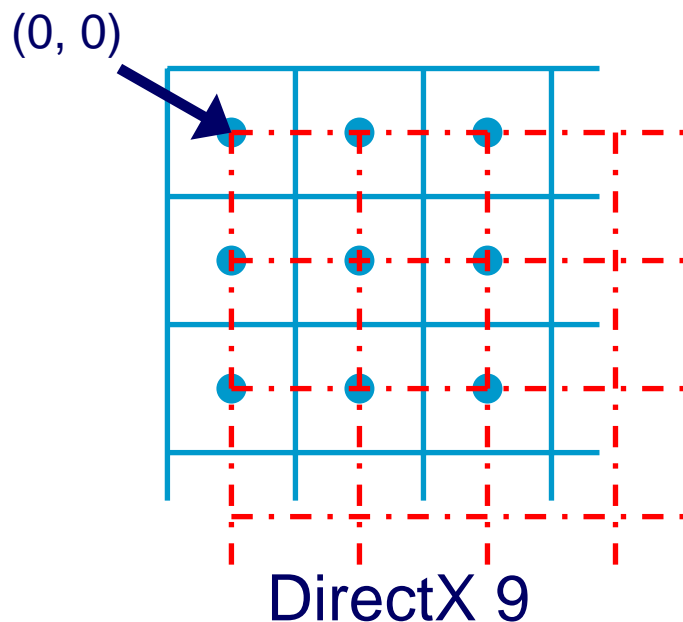
```
DXGI_SWAP_CHAIN_DESC scd;  
scd.Flags = DXGI_SWAP_CHAIN_FLAG_ALLOW_MODE_SWITCH;
```

```
D3D10CreateDeviceAndSwapChain(..., scd, ...);
```

- ⊕ Otherwise back buffer is stretch blit'ed to desktop resolution

Pixel Coordinate System

- ⊕ Finally pixels and texels match
- ⊕ Don't need to offset position or texture coords by 0.5 texel





Pixel Coordinate System

- ⌚ This will affect all the screen space DirectX 9 code/shaders
- ⌚ If offset is done in PS, move it to app code
 - ⌚ Make shaders DX9/10 cross-compatible



Small Batch Problem

- ⌚ DirectX 10 API helps
 - ⌚ Currently up to 2x measured improvement over DX9 from API changes
- ⌚ New features provide additional boost
 - ⌚ Instancing
 - ⌚ Uber-shaders
 - ⌚ GS (e.g.. render to cubemap)
- ⌚ Trade GPU performance to solve problem



Instancing

- ⌚ Draw as many objects per draw call as possible
- ⌚ Create object variations with...
 - ⌚ Large constant storage
 - ⌚ Displacement maps
 - ⌚ Uber-shaders



Uber-shaders

- ④ Combine multiple materials within single shader
- ④ Pros:
 - ④ Keep common code portion in on-chip shader cache
- ④ Cons:
 - ④ Flow control
 - ④ Higher GPR pressure



Optional DX10 Features

- ④ Yes, there's such a thing
 - ④ Some format support is optional
 - ④ E.g. MSAA
 - ④ E.g. FP32 filtering
- ④ Always check with
`ID3D10Device::CheckFormatSupport()`



MSAA Capabilities

- ④ Multiple support flags for MSAA:
 - ④ D3D10_FORMAT_SUPPORT_MULTISAMPLE_RENDER_TARGET
 - ④ D3D10_FORMAT_SUPPORT_MULTISAMPLE_RESOLVE
 - ④ D3D10_FORMAT_SUPPORT_MULTISAMPLE_LOAD
- ④ Some formats might be renderable, but not resolvable!



Constants

- ⌚ Literal HLSL constants are the fastest, don't put common constants in buffers
 - ⌚ E.g. -1, 0.5, 2...
- ⌚ Non-indexed constants or indexed with a literal could be faster than indexed with a computed value
 - ⌚ Is there anything you can do to help this, beyond simple unrolling?



Constant Management

- ⌚ Need to specify const location in HLSL when not using Effects
- ⌚ Even with Effects might want to do “smarter” custom const management
- ⌚ Special syntax for “manual” constant placement



Manual Const Binding

⦿ HLSL syntax

```
// Will bind constant buffer to slot #4
cbuffer MyConstantBuffer : register(b4)
{
    // Specify exact offset
    float  fMyConstant1 :   packoffset(c2.z);
    float4 vecMyConstant2 : packoffset(c4);
    // etc.
};
```



Primitive Topology and Draw Calls

- ③ Primitive topology is separate from draw function

```
dev->IASetPrimitiveTopology(blah);  
dev->Draw(3, 0);
```

- ③ Keep these 2 functions together
 - ③ Easy to forget to set proper topology
 - ③ Will be chasing false corruption, driver bug and etc.



Input Layout

- ④ Matching vertex data to VS inputs
- ④ Unlike in DirectX 9 it now requires knowledge about VS (input signature)
- ④ Don't need to create a unique one for each VS
 - ④ Enough if shader signature matches



Input Layout

- ⌚ Knowledge of VS at decl/input layout creation time might be a problem for DX9 engine architectures
- ⌚ Quick DX9 port hack
 - ⌚ Keep copies of “dummy” VS just for the signatures
 - ⌚ Create input layouts from them



Stream Out

- ④ Primitive topology is converted to lists
- ④ For subsequent passes loses the benefit of the vertex reuse
- ④ Solution:
 - ④ Use points for stream out
 - ④ Use indexed primitives with proper topology on the second pass



Stream Out

- ⌚ With SO enabled each call appends to the end of the bound buffers
- ⌚ Need to bind the same buffers again to output to the beginning of the buffers
 - ⌚ Might interfere with renderer state caching
- ⌚ Good idea to un-bind buffers as soon as done with SO
 - ⌚ Easy to forget and cause some corruption



Stream Out FX Syntax

```
// Output position only  
SetGeometryShader(ConstructGSWithSO(shader,  
    "SV_Position"));
```

```
// Output 2 streams, use 2D tex coords  
SetGeometryShader(ConstructGSWithSO(shader,  
    "0:SV_Position,1:tex.xy"));
```

```
// Output only z component of position  
SetGeometryShader(ConstructGSWithSO(shader,  
    "SV_Position.z"));
```



Stream Out From VS

- ⌚ Output is after GS, but GS can be pass-through NULL shader
 - ⌚ Pass VS code instead of GS
 - ⌚ Just requires the output signature from VS
- ⌚ Don't write the actual GS shader!



Stream Out From VS

⊕ Effects code:

```
technique10 t0
{
    pass p0
    {
        SetVertexShader(CompileShader(vs_4_0,
            VsMain()));
        SetGeometryShader(ConstructGSWithSO(
            CompileShader(vs_4_0, VsMain()),
            "SV_Position"));
        SetPixelShader(CompileShader(ps_4_0, PsMain()));
    }
}
```




SO as VS optimization

- ③ Unified architecture requires a new thinking
 - ③ VS isn't free anymore when PS-bound
 - ③ Think about total workload throughout VS/GS/PS
- ③ Could stream out data that would be computed multiple times in VS/GS
 - ③ E.g. animation
- ③ SO might not be a win for small VS



Scatter Implementation

- ⌚ SO isn't flexible enough to implement scatter
- ⌚ Could implement scatter with point primitives
 - ⌚ Tweak point position in VS/GS
- ⌚ E.g. sort on GPU, data binning, etc.



Data Binning with Scatter

- ④ E.g. building histogram for HDR
 - ④ Draw points
 - ④ Output is 1D render target with histogram
 - ④ For each point in the image
 - ④ Fetch from RTT in VS
 - ④ Adjust position (bin index) based on the texture value, output 1 for counting
 - ④ Additively blend to count points in the bins



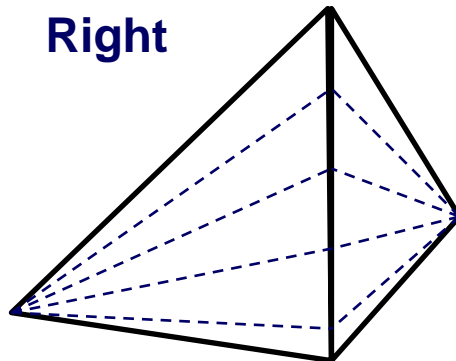
Shader Signature Matching

- ⌚ In DirectX 9 semantics matched automatically
- ⌚ In DirectX 10 shader input/output structure order should match
 - ⌚ Put optional data at the end
 - ⌚ Partial match might be hard to track
- ⌚ Might be a big problem for DX9 ports

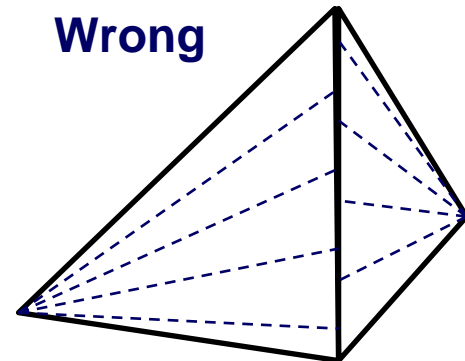
GS: Edge Orientation

- ⌚ When computing per-edge data need to ensure it matches for adjacent triangles
 - ⌚ Could tag vertices for selecting proper orientation
- ⌚ E.g. edge tessellation, fur fins

Right

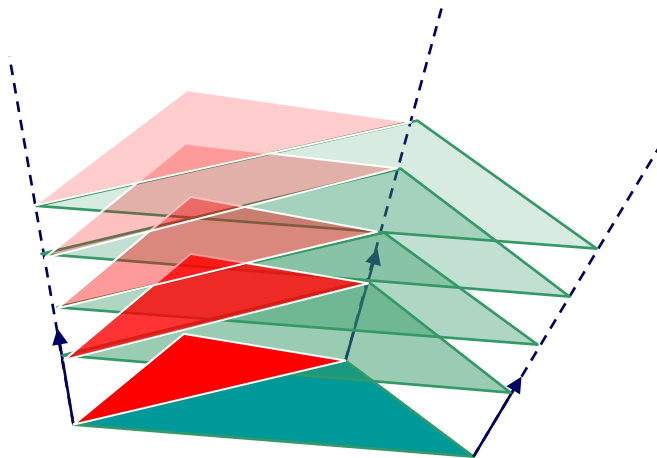


Wrong



GS: Order of Output Triangles

- ⌚ Well defined output triangle order
 - ⌚ It's as if each triangle processed serially
- ⌚ Consider interaction with transparency
 - ⌚ E.g. won't work for fur shells generation





GS: Triangle Winding

- ⌚ Backface culling happens in the rasterizer after GS
- ⌚ Need to keep winding in mind when generating geometry in GS
 - ⌚ Easy to neglect and blame drivers for missing geometry



GS: Render to Cubemap

- ④ An interesting feature to combat small batch performance
 - ④ GS replicates triangles to different cubemap faces
- ④ Performance tradeoff
 - ④ Lighter CPU load (fewer draw calls)
 - ④ Heavier GPU load
- ④ Could cull in GS to reduce amount of generated data



Render to Cubemap Example 1/2

```
[maxvertexcount(18)]
void main(triangle GsInShadow In[3],
  inout TriangleStream<PsInShadow> Stream)
{
  PsInShadow Out;
  // Loop though all faces
  [unroll]
  for (int k = 0; k < 6; k++) {
    // Select face target
    Out.target = k;
    // Transform verts
    float4 pos[3];
    pos[0] = mul(mvpArray[k], In[0].pos);
    pos[1] = mul(mvpArray[k], In[1].pos);
    pos[2] = mul(mvpArray[k], In[2].pos);
    // Frustum culling
    float4 t0 = saturate(pos[0].xyxy*float4(-1,-1,1,1)-pos[0].w);
    float4 t1 = saturate(pos[1].xyxy*float4(-1,-1,1,1)-pos[1].w);
    float4 t2 = saturate(pos[2].xyxy*float4(-1,-1,1,1)-pos[2].w);
    float4 t = t0 * t1 * t2;
    [branch]
    if (!any(t)) {
```



Render to Cubemap Example 2/2

```
. . .
    // Back face culling
    float2 d0 = pos[1].xy/abs(pos[1].w)-pos[0].xy/abs(pos[0].w);
    float2 d1 = pos[2].xy/abs(pos[2].w)-pos[0].xy/abs(pos[0].w);
    [branch]
    if (d1.x * d0.y > d0.x * d1.y) {
        // Triangle is visible - emit
        [unroll]
        for (int i = 0; i < 3; i++) {
            Out.pos = pos[i];
            // Other data processed here
            // . . .

            Stream.Append(Out);
        }
        Stream.RestartStrip();
    }
}
}
```



Position in PS

- ⌚ PS could have SV_Position as input
 - ⌚ ...VS or GS also have SV_Position output
- ⌚ Same name, different values
 - ⌚ VS/GS – position in clipping space
 - ⌚ PS – screen space position (like vPos register) and z, rhw
- ⌚ Returns 0.5, 1.5, 2.5, ... coordinates
 - ⌚ Due to new pixel coordinate system



Integer Data and Instructions

- ④ Use integer types to compact data
 - ④ Bit packing vertex data, consts, etc.
 - ④ E.g. conditionals for uber-shaders encoded per vertex or per primitive
- ④ Beware of some int instruction cost
 - ④ Division is expensive
 - ④ Explore optimization opportunities



Backwards Compatible Shader Compilation

- ⌚ Special compiler flag
 - ⌚ `D3D10_SHADER_ENABLE_BACKWARDS_COMPATIBILITY`
 - ⌚ Enable old shaders to compile to SM 4.0
 - ⌚ Not valid for geometry shaders
- ⌚ Could cause compilation errors when mixing GS with old VS
 - ⌚ Old `POSITION` semantic is translated to `SV_Position`
 - ⌚ Solution: replicate structures for GS with new semantics



Texture/Sampler States

- ⌚ Samplers and textures are decoupled in DirectX 10
- ⌚ Important to keep in mind when porting from DirectX 9
 - ⌚ Might require significant engine re-architecture effort



Disabling Mipmapping

- ⌚ There's no NONE mip filtering mode in DirectX 10
 - ⌚ No direct way to disable mipmapping
- ⌚ Hacks to emulate functionality
 - ⌚ Textures with only 1 mip level
 - ⌚ Setting MaxLOD in sampler state to 0

```
D3D10_SAMPLER_DESC samp;  
samp.MaxLOD = 0.0f;
```



Integer Textures

- ⌚ Default texture declaration type is float
- ⌚ Integer type texture declaration

```
// Using unsigned int
Texture2D <uint4> myTex0;
// Using int
Texture2D <int4> myTex1;
```

- ⌚ E.g. important when reading stencil format (X24_TYPELESS_G8_UINT)



Reading Depth/Stencil

- ⌚ Cannot simultaneously read depth and stencil from the same buffer
 - ⌚ Create 2 separate shader resource views
 - ⌚ Depth
 - ⌚ Stencil
 - ⌚ Treat as 2 separate textures in shader
- ⌚ MSAA depth/stencil buffers can't be read



Gradients and Flow Control

- ⌚ Same as before, can't sample textures with varying texture coordinates (different across pixel quad)
- ⌚ Compiler is smart figuring out what is varying and what isn't
 - ⌚ Don't blindly use `t.SampleGrad()` everywhere



MRT Rendering

- ⌘ MRTs are more flexible in DirectX 10
 - ⌘ Up to 8 render targets
 - ⌘ Any of 8 slots can be set
- ⌘ For performance reasons use the lowest MRT slots possible
 - ⌘ Don't leave holes in MRT slot assignment
- ⌘ Can't mix MSAA and non-MSAA



MRT Rendering

- ⌚ Even if MRT #0 isn't bound, target #0 alpha is used for Alpha-to-Coverage
- ⌚ Don't forget to enable render target masks to enable MRT output
 - ⌚ Separate controls for each MRT

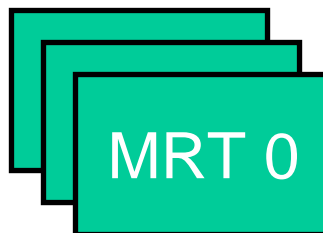
```
// enable MRT
D3D10_BLEND_DESC bd;
bd.RenderTargetWriteMask[0] = 0x0f;
bd.RenderTargetWriteMask[1] = 0x0f;
bd.RenderTargetWriteMask[4] = 0x0f;
```

MRT vs. Indexable Render Target Array

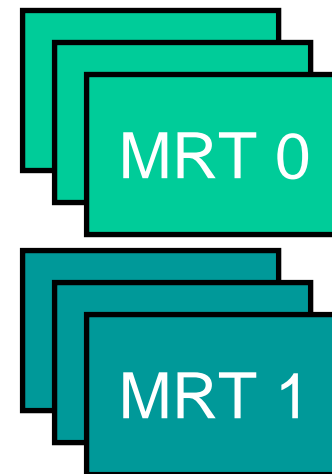
- ⦿ MRT and render target arrays are orthogonal



MRT example



**Render array
example**



**MRT + render array
example**



NULL Outputs

- ⌚ Can have NULL SO, RTs and depth
- ⌚ Can disable RTs/depth while doing stream out only



Depth Clamping/Clipping

- ⌚ Back-end always clamps depth
 - ⌚ Both interpolated and from PS
- ⌚ Viewport can enable/disable depth clipping before it gets to clamping

```
// enable depth clipping  
D3D10_RASTERIZER_DESC rd;  
rd.DepthClipEnable = true;
```

- ⌚ Also, $W < 0$ will clip



sRGB implementation differences

- ④ DirectX 10 sRGB implementation is different from DirectX 9
 - ④ Differently spec'ed gamma curve
 - ④ All blend and filter in linear space
 - ④ sRGB fetch – degamma before filter
 - ④ sRGB blend – degamma DEST before blend
- ④ DirectX 10: correct implementation that could make DX9 content look wrong



Separate Alpha Blend

- ⌘ No more separate alpha blend enable
- ⌘ It's always on, so don't forget to set
 - ⌘ SrcBlendAlpha
 - ⌘ DestBlendAlpha
 - ⌘ BlendOpAlpha



Dual Source Color Blending

- ⌚ Uses 2 PS outputs for blending equation
- ⌚ New alpha blend arguments
 - ⌚ D3D10_BLEND_SRC1COLOR
 - ⌚ D3D10_BLEND_INV_SRC1COLOR
 - ⌚ D3D10_BLEND_SRC1ALPHA
 - ⌚ D3D10_BLEND_INV_SRC1ALPHA
- ⌚ Doesn't work with MRT!



Alpha-to-Coverage

- ⌚ Works even without MSAA
 - ⌚ Can produces screen-door effect
- ⌚ Implementation is IHV dependent
 - ⌚ Grey area of the spec
 - ⌚ For better quantization HW can use some area dithering
 - ⌚ Be careful not to make assumptions about implementation



Custom AA Resolves

- ④ Application can implement custom resolves in PS
 - ④ Access to individual RT samples
 - ④ Special syntax for accessing samples

```
// Declare MSAA texture with 4 samples
Texture2DMS<float4,4> t;
// Load sample #0
a = t.Load(tc, 0); // tc - unnormalized tex. coords
// Load sample #1
b = t.Load(tc, 1);
```

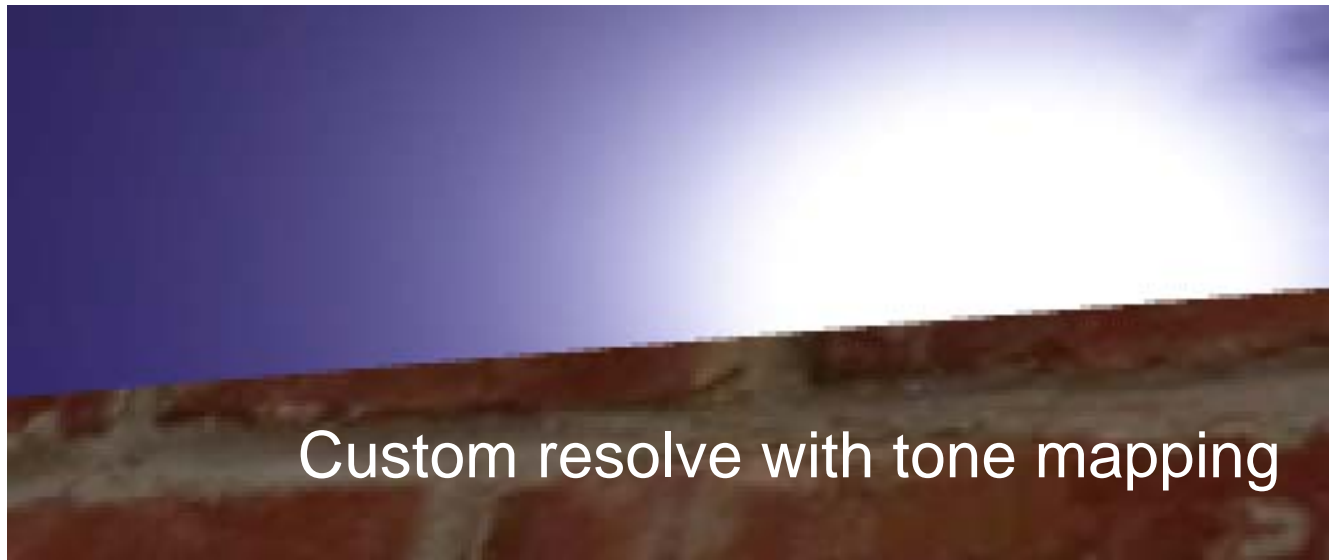
Custom AA Resolves and HDR

- ⦿ Need to perform tone mapping before resolve to get correct MSAA results



Custom AA Resolves and HDR

- ⦿ Need to perform tone mapping before resolve to get correct MSAA results





Custom AA Resolves and HDR

```
Texture2DMS<float4, SAMPLES> tHDR;
```

```
float4 main(float4 pos: SV_Position) : SV_Target
{
    int3 coord;
    coord.xy = (int2)In.pos.xy;
    coord.z = 0;
    // Correct exposure for individual samples and sum it up
    float4 sum = 0;
    [unroll]
    for (int i = 0; i < SAMPLES; i++)
    {
        float4 c = tHDR.Load(coord, i);
        sum.rgb += 1.0 - exp(-exposure * c.rgb);
    }
    sum *= (1.0 / SAMPLES);
    // Gamma correction
    sum.rgb = pow(sum.rgb, 1.0 / 2.2);
    return sum;
}
```



Acknowledgement & References

- ⊕ Big thanks for help and comments to:
 - ⊕ Nicolas, Emil, Natasha, Thorsten and the rest of ISV and 3DARG teams
- ⊕ Real-time HDR histogram generation
 - ⊕ Check out upcoming I3D '07 paper by Thorsten Scheuermann



Questions